

UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL RABELO ITAJUBÁ

THINK: GUIA DE DESENVOLVIMENTO MOBILE ÁGIL EM
EQUIPES REDUZIDAS

BAURU

2013

Rafael Rabelo Itajubá

THINK: GUIA DE DESENVOLVIMENTO MOBILE ÁGIL EM EQUIPES REDUZIDAS

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, Unesp, campus de Bauru, como parte do Trabalho de Conclusão de Curso.

Orientador: Prof. Dr. Eduardo Martins Morgado

Co-orientador: Lais Munhoz Mastelari

BAURU

2013

Rafael Rabelo Itajubá

THINK: GUIA DE DESENVOLVIMENTO MOBILE ÁGIL EM
EQUIPES REDUZIDAS

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, Unesp, campus de Bauru, como parte do Trabalho de Conclusão de Curso.

BANCA EXAMINADORA

Prof. Dr. Eduardo Martins Morgado
Professor Doutor
DCo - FC - UNESP - Bauru
Orientador

Prof. Dr. Simone Domingues Prado
Professora Doutora
UNESP – Bauru

Prof. Dr. Sidnei Bergamaschi
Professor Doutor
UNESP – Bauru

Bauru, 22 de Novembro de 2013.

AGRADECIMENTOS

Em primeiro lugar, eu gostaria de agradecer à Lais Mastelari, de quem eu tirei energia, força e inspiração não só para terminar este trabalho, como também para fazer tudo o que fiz até hoje.

Também agradeço imensamente à minha mãe, Elânia Maria de Freitas Rabelo por me incentivar nos momentos de dificuldade e me aturar nos momentos de estresse.

Agradeço também ao Prof. Morgado, que me apoiou em vários momentos importantes e me mostrou o caminho todas as vezes que me perdi, sendo muito mais do que um orientador, e sim um mentor e amigo. Sem ele e o LTIA, minha experiência na faculdade com certeza teria sido menos rica e construtiva. Aproveito também para agradecer à todos do LTIA que fizeram parte da minha vida diariamente, me ajudando a descobrir coisas novas e maravilhosas todos os dias.

Por último, mas não menos importante, agradeço a todos meus professores e colegas da faculdade pela ótima educação e pelos bons momentos que tivemos juntos, dos quais jamais esquecerei.

RESUMO

Atualmente, aplicações para *smartphones* e *tablets*, chamadas apps, estão se tornando cada vez mais relevantes e atraem mais atenção dos usuários e, por fim, dos desenvolvedores. Com as Lojas de Aplicativos, serviços disponibilizados pela empresa que sustenta a plataforma, o acesso a tais aplicações está tão ou mais simplificado que aos sites web, com a vantagem de uma experiência do usuário aprimorada e voltada ao dispositivo mobile, e de usufruir de recursos nativos como câmera, áudio, armazenamento, integração com outras aplicações, etc.

Elas apresentam uma grande oportunidade para desenvolvedores independentes, que agora podem produzir uma aplicação e disponibiliza-la a todos os usuários daquela plataforma, gratuitamente ou a um custo que costuma ser baixo. Até mesmo estudantes podem criar suas aplicações nos intervalos de suas aulas e vende-las nas lojas.

Fazendo uso de ferramentas e serviços, gratuitos ou de baixo custo, é possível desenvolver aplicações de qualidade que podem ser comercializadas e ter um grande número de usuários, mesmo em situações adversas no qual a aplicação não é o foco da produtividade do desenvolvedor.

Porém, tais ferramentas não parecem ser bem aproveitadas, ou são desconhecidas, ou seu propósito não é considerado importante, e este trabalho tenta mostrar a real importância destas ferramentas no desenvolvimento rápido de software de qualidade.

Este projeto apresenta inúmeras ferramentas, serviços e práticas, que em conjunto, tornam possível desenvolver um aplicativo, para diversas plataformas *mobile*, de maneira independente e com uma equipe de poucas pessoas, como é demonstrado.

Porém este trabalho não visa dizer que o desenvolvimento de *software*, nos dias de hoje, é fácil e simples, e sim que atualmente existem um conjunto enorme de ferramentas, para diversas plataformas, que auxilia e otimiza o trabalho do programador.

Palavras-chave: Engenharia de Software, Aplicações Mobile, Multiplataforma, Ferramentas, Processo de Desenvolvimento de Software, Empreendedorismo, Scrum, Lean.

ABSTRACT

Currently, applications for smartphones and tablets, called apps, are becoming increasingly relevant and attract more attention from users and finally the developers. With the Application Stores, services provided by the company that maintains the platform, access to such applications is as or more simplified than to web sites, with the advantage of an enhanced user experience and focused on the mobile device, and enjoy natives resources as camera, audio, storage, integration with other applications, etc.

They present a great opportunity for independent developers, who can now develop an application and make it available to all users of that platform, at free or at a cost that is usually low. Even students may create their applications in the intervals of their classes and sell them in stores.

Making use of tools and services, free or at low cost, anyone can develop quality applications, that can be marketed and have a large number of users even in adverse situations in which the application is not the focus of developer productivity.

However, such tools do not seem to be well used, or are unknown, or its purpose is not considered important, and this paper tries to show the real importance of these tools in the rapid development of quality software.

This project presents several tools, services and practices, which together make it possible to develop an application for various mobile platforms, independently and with a team of a few people, as demonstrated.

However, this paper aims not to say that the development of software today it is easy and simple, but there are currently a large set of tools, for various platforms, that assists and enhances the work of the programmer.

Keywords: Software Engineering, Mobile Applications, Multi-platform, Tools, Software Development Process, Entrepreneurship, Scrum, Lean.

LISTA DE FIGURAS

Figura 1 - Exemplo de Etapas de Desenvolvimento utilizando Servidor Remoto.....	16
Figura 2 - Utilização de Arquitetura Web Services REST e modelo Software-as-a-Service ..	17
Figura 3 - Manifesto do Software Ágil	18
Figura 4 - Organização das Etapas do Processo	28
Figura 5 - Exemplo de Organização das Etapas do Processo	28
Figura 6 - Processos orientados a Ciclos	29
Figura 7 - Think na Windows Store	30
Figura 8 - Exemplo do versionamento nas etapas de desenvolvimento	36
Figura 9 - A aplicação de esforço nas características de um projeto de Software	37
Figura 10 - Think para Android - Smartphone	39
Figura 11 - Think para Android - Tablet	40
Figura 12 - Think para Windows Phone, MVP	40
Figura 13 - Think para Windows	41
Figura 14 - API REST	41
Figura 15 - Servidor rodando Arch Linux	42
Figura 16 - Ambiente virtualizado de Homologação	43

SUMÁRIO

1	INTRODUÇÃO.....	9
1.1	PROBLEMA.....	10
1.2	JUSTIFICATIVA	11
1.3	OBJETIVOS	12
1.3.1	Objetivo Geral	12
1.3.2	Objetivos Específicos	12
2	FUNDAMENTAÇÃO TEÓRICA	13
2.1	AMBIENTES	13
2.2	FRAMEWORKS.....	14
2.3	ETAPAS DE DESENVOLVIMENTO	15
2.4	MODELO SOFTWARE-AS-A-SERVICE E ARQUITETURA REST.....	16
2.5	DESENVOLVIMENTO ÁGIL.....	18
2.6	DESENVOLVIMENTO MOBILE	19
2.6.1	Mobile-D	20
2.7	METODOLOGIA LEAN.....	20
2.8	EXPERIMENTAÇÃO E EXEMPLIFICAÇÃO NA ENGENHARIA DE SOFTWARE	21
3	FERRAMENTAS.....	23
3.1	AMBIENTE DE DESENVOLVIMENTO.....	23
3.2	AMBIENTE DE HOMOLOGAÇÃO	24
3.3	AMBIENTE DE PRODUÇÃO.....	25
3.4	FRAMEWORKS E LIBRARY	25
4	RESULTADOS	27
4.1	CARACTERÍSTICAS DO PROCESSO DE DESENVOLVIMENTO.....	27
4.2	EMPREENHIMENTO: A IDEIA	30
4.2.1	Multiplataforma.....	31
4.3	GERENCIA DO PROJETO: A ORGANIZAÇÃO	32
4.3.1	Scrum.M Reduzido	34
4.4	DESENVOLVIMENTO: A PRODUÇÃO	35
4.4.1	Etapas de Desenvolvimento	35
4.4.2	Aplicação de Esforço	37
4.4.3	Organização de Código	38
4.5	APLICAÇÕES E PROJETOS	39
4.6	GUIA DE DESENVOLVIMENTO	44
5	CONCLUSÃO	45
	REFERÊNCIAS	46

1 INTRODUÇÃO

Atualmente é simples desenvolver um software. O investimento necessário inicial são alguns dólares para uma conta nas lojas de aplicativo para smartphone, como a Play Store do Google para Android, ou a App Store da Apple para iOS. Ou ainda, com o advento dos *softwares* web hospedados em serviços de Cloud Computing, como o Azure da Microsoft, acrescentando mais alguns dólares é possível criar um sistema web complementemente escalável para este aplicativo, transformando-o em um serviço multiplataforma, utilizando menos de 200 dólares iniciais.

Soma-se isso aos flexíveis modelos de gerência de projetos existentes, como Scrum, Kanban ou Extreme Programming, todos voltados para pequenas equipes e ao grande conjunto de ferramentas de desenvolvimento profissionais gratuitas disponíveis tornam possível a criação de sistemas elaborados e completos, e ainda monetizá-los, mesmo sem pertencer a uma empresa ou possuir um grupo grande de pessoas para o projeto. É possível desenvolver e publicar um software simples com uma única pessoa.

Porém, conforme a complexidade da aplicação aumenta técnicas de engenharia de software e processos de desenvolvimento formais se tornam necessários, mas raramente são empregados (WASSERMAN, 2010). Torna-se preciso não somente produzir o código, mas gerir o código, realizar testes, mantê-lo modular e no caso da plataforma mobile, apto às mudanças necessárias do mercado, necessidade de operar em várias plataformas e necessidades específicas do usuário mobile final (ABRAHAMSSON *et al.*, 2004). Para um grupo pequeno de desenvolvedores independentes iniciantes tais questões podem ser desmotivadoras uma vez que não são abordadas com profundidade nas instituições de ensino.

Este projeto de conclusão visa criar um guia que será elaborado contendo as boas práticas, os métodos utilizados e as experiências de desenvolver para cada plataforma proposta, para que um estudante possa seguir tais práticas e conseguir desenvolver seu aplicativo em um roteiro de estudo simplificado. Além disso, também será necessária a criação de aplicativos usando métodos de desenvolvimento e de gerência a serem estudados e explicados, para exemplificar que tal estudo permite que o ciclo de produção seja possível mesmo com uma equipe reduzida.

O projeto também abordará a questão do desenvolvimento rápido utilizando tanto ferramentas de automatização de processos diversos, como testes, quanto a *frameworks* para minimizar o tempo de desenvolvimento das aplicações e do sistema em si. Este será um ponto

importante, pois estes projetos raramente possuem investimento externo e mesmo quando possuem, necessitam mostrar resultados o quanto antes para não perderem valor e credibilidade (FAYAD, LAITINEN e WARD, 2000).

Também dada à natureza comercial do sistema a ser desenvolvido e da monetização dos aplicativos, serão abordadas brevemente questões de empreendedorismo e boas práticas para manter aplicativos desenvolvidos sob o guia proposto rentável e saudável financeiramente.

“Think” é o sistema que será desenvolvido em conjunto com o guia e alvo do estudo deste trabalho. Sua equipe de desenvolvimento será Rafael Rabelo Itajubá, programador e autor deste trabalho, e Lais Munhoz Mastelari, designer. Tal sistema será um simples aplicativo para compartilhar citações e frases de autores famosos.

1.1 Problema

As instituições de ensino da área de desenvolvimento de software focam seus materiais e suas disciplinas no estudo dos algoritmos e da produção de código, tornando o discente um bom programador. Este, porém, ao chegar ao mercado de trabalho se vê em uma situação diferente do que foi aprendido na faculdade com questões de desenvolvimento, como versionamento, escalabilidade, testes unitários, modularização, compatibilidade entre diferentes versões, desenvolvimento multiplataforma, integração e uma série de pontos que, dependendo do contexto do projeto, são tão importantes quanto à produção do código em si.

Pesquisas mostram (SBC, 2006) os problemas da falta de qualidade em software e consideram suas implicações sobre a indústria nacional. Deve existir a preocupação com o desenvolvimento de software em equipe, de forma distribuída, dinâmica, com aplicação de engenharia e baseado em evidência ao longo das disciplinas relacionadas à Engenharia de Software. Isso atinge diretamente o desafio desenvolvimento tecnológico de qualidade. (SANTOS *et al.*, 2008)

Porém, a partir do ponto de vista do contexto deste projeto, no qual jovens empreendedores podem desenvolver suas aplicações com pouco capital inicial e publicá-las independentemente, a falta deste conhecimento e experiência de desenvolver aplicações completas e concisas é desmotivadora e o maior obstáculo para estes estudantes. Sem isso, ele pode ficar frustrado ao tentar desenvolver um aplicativo e perceber que seus conhecimentos adquiridos durante a faculdade não resolveram questões cruciais em relação à produção de um aplicativo real e pronto para ser publicado, fazendo com que ele desista (DE LUCENA, BRITO e GOHNER, 2006)

Tal comportamento inibe, portanto, a inovação tecnológica que os jovens podem trazer quando se trata de software, principalmente mobile, que facilita o desenvolvimento de aplicações independentes.

1.2 Justificativa

A área de desenvolvimento de *software* é um ramo diferenciado na indústria da tecnologia, pois não necessita de matéria prima para sua produção e os custos com sua distribuição são extremamente baixos com a ascensão da internet e conectividade. Logo, países em desenvolvimento da base da pirâmide podem usufruir desta característica para aumentar a inovação e, portanto, sua competitividade (PRAHALAD e HART, 2001). Além disso, estes mesmos países podem, com aplicativos a baixo custo, se aproveitar das características de seu mercado e se focarem em quantidade de vendas e downloads (PRAHALAD e HART, 2001).

Este projeto, com seu guia e seu estudo de caso, tem por principal justificativa tornar mais fácil, simples e documentado o processo de criar um aplicativo comercial desenvolvido independente e informalmente, porém de maneira séria e funcional tecnicamente, e assim estimular desenvolvedores a criar suas próprias aplicações e sistemas, e assim aumentar a inovação tecnológica no país.

Um estudo sobre métodos de desenvolvimento poderá ser útil aos que têm interesse em começar seu próprio negócio após terminar a faculdade, aos interessados em desenvolver aplicações em pequenos grupos e também aos professores que ensinam sobre Engenharia de Software. Estes possuiriam um guia para desenvolvimento com lista de ferramentas e métodos, além de um estudo de caso, que melhorariam o ensino da disciplina (SANTOS *et al.*, 2008).

Também estimularia desenvolvedores independentes a seguirem práticas consideradas saudáveis para a manutenção de um projeto de software além de lançar novas perspectivas sobre aqueles que estão neste caminho, porém não possuem um guia centralizado e estruturado para comparar suas próprias experiências.

1.3 Objetivos

1.3.1 Objetivo Geral

Redigir um guia sobre métodos de desenvolvimento de software e gerência de projeto que sejam úteis aos desenvolvedores independentes e/ou que trabalhem em equipes reduzidas.

1.3.2 Objetivos Específicos

- Aprender sobre diferentes tipos de gerência de projetos, que sejam úteis às pequenas equipes, como Scrum e Kanban.
- Aprender sobre métodos de desenvolvimento de *software*, que utilizem Controle de Versão, ferramentas de *deployment* como Maven e de Integração Continuada como Hudson, para Java EE e seus respectivos para desenvolvimento em C#.
- Modificar estes métodos para que sejam condizentes com a realidade do desenvolvimento em equipes reduzidas.
- Realizar um levantamento de ferramentas necessárias para a configuração do ambiente do método de desenvolvimento, com IDEs utilizadas, *plugins*, *softwares* e ferramentas *standalones*, servidores, serviços e infraestrutura.
- Publicar os aplicativos desenvolvidos e continuar com o ciclo de vida dos aplicativos, realizando um estudo estatístico das ações e seus impactos na popularidade do mesmo;
- Redigir o Guia de Desenvolvimento em equipes reduzidas;

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Ambientes

Um ambiente de desenvolvimento é um conjunto de ferramentas e processos para desenvolver, testar e validar uma aplicação ou programa. Este está normalmente ligado à três configurações diferentes, desenvolvimento, homologação (ou *staging*¹) e produção. Estas três configurações são, normalmente, utilizadas em conjunto pelo(s) desenvolvedor(es).

- Configuração de Desenvolvimento: Acessada diretamente pelo desenvolvedor, isolada do acesso pelos clientes, pois provavelmente conterá erros e funções não implementadas. Pode ser criada localmente ou um servidor para ser acessada por vários desenvolvedores simultaneamente.
- Configuração de Homologação: Esta configuração tende a ser o mais parecida possível com a configuração de produção, ou seja, se trata de como a aplicação se comportaria em um ambiente próximo do real, utilizado pelo cliente.
- Configuração de Produção: A última etapa e mais crucial pois se trata da publicação do software aos clientes. O software somente é aplicado nesta configuração se não apresentou nenhum defeito considerado relevante.

Este termo também pode ser utilizado como sinônimo de IDE ou Ambiente de Desenvolvimento Integrado, que se refere a uma ferramenta utilizada para escrever, testar e compilar programas e também proveem ao desenvolver uma Interface de Usuário, que pode variar de acordo com o modo em que esta ferramenta se encontra.

Porém, “ambiente de desenvolvimento” se refere a um conjunto de processos maior do que as IDEs podem oferecer de maneira integrada pois existem determinados sistemas que utilizam serviços remotos que não podem ser integrados diretamente nas IDEs e que para serem acessados necessitam de plug-ins integrados às IDEs, e quando isto não é possível, ferramentas específicas.

Por exemplo, existem quatro grandes conjuntos de ferramentas e serviços que podem ser utilizadas em um ambiente de desenvolvimento:

¹ Ambiente de teste similar ao ambiente de produção

- IDEs: Utilizadas localmente pelos desenvolvedores para produzir, testar e gerir o código fonte e arquivos compilados.
- Controle de Versão: Utilizado por desenvolvedores para gerir o código-fonte produzido local e remotamente, evitando assim a perda de código e facilitando o compartilhamento deste entre vários desenvolvedores de um time.
- Acompanhamento de Tarefas (*Issue Tracker*): Permite rastrear defeitos e requisições de novas funcionalidades tanto pelos clientes quanto pelos desenvolvedores e *stakeholders*².
- Integração Continuada: Ferramentas utilizadas para testar o *software* em vários níveis e, dependendo de sua configuração, podendo controlar qual Configuração de Desenvolvimento o *software* se encontra.

Estas três últimas ferramentas normalmente são utilizadas remotamente, de maneira que vários desenvolvedores possam compartilhar informações através delas, seja código, funcionalidades e bugs ou informações a respeito de testes e do estágio de desenvolvimento em que o *software* se encontra.

2.2 Frameworks

Frameworks são “uma biblioteca de classes que captura padrões de interação entre objetos e consiste em um conjunto de classes concretas e abstratas explicitamente projetadas para serem utilizadas em conjunto. Aplicações são desenvolvidas a partir de um *framework* a partir da complementação e implementação de classes abstratas.” (ROGERS, 1997) e portanto estes são parte integrante, se não essencial, de um *software* que os utiliza, diferentemente de ferramentas do ambiente de desenvolvimento. Porém, dada sua importância para o projeto e para o resultado final, a escolha de *frameworks* é considerada tão essencial quanto a escolha da linguagem de desenvolvimento, tendo em vista que este deve suprir a maior quantidade possível de funcionalidades afim de maximizar a reutilização de código e, ao mesmo tempo, ser eficiente.

Apesar da evidência prática sugerir que o uso frameworks pode aumentar a reusabilidade e diminuir o esforço de desenvolvimento (MOSER e NIERSTRASZ, 1996), a experimentação

² Grupo de pessoas ou organizações integrantes ou não do projeto que estejam envolvidas no mesmo.

tem identificado um número de questões que complicam a aplicação de *frameworks* e limita seus potenciais benefícios (BOSCH, MOLIN, *et al.*, 1999).

Sendo assim, a escolha dos frameworks deve ser feita levando em consideração diversos testes e estudos. Todos os frameworks escolhidos para o desenvolvimento deste projeto são de código aberto e possuem casos de teste bem elaborados e validados, atestando uma qualidade elevada.

2.3 Etapas de Desenvolvimento

Independentemente do método utilizado para gerir o projeto, o desenvolvimento em si pode ser dividido em processos menores ou etapas de desenvolvimento. Tais etapas estão ligadas às ferramentas disponíveis no ambiente de desenvolvimento pois certos processos, como dito anteriormente, não podem ser realizados sem as ferramentas e sistemas corretos.

Por exemplo, no método Scrum de gerência de projetos, o termo Sprint é utilizado para se referir a uma versão de *software* que foi testada e que atende uma série de Stories³ pré-determinadas. Porém, nada é dito sobre como serão realizados os Testes Unitários e Testes de Integração, ou quais parâmetros serão utilizados para os desenvolvedores compartilharem as alterações em seus códigos, pois isto é dependente do ambiente de desenvolvimento estabelecido pela equipe que executará o projeto e garantirá sua qualidade em nível técnico.

Pode-se estabelecer, portanto, um ciclo de desenvolvimento, composto pelo conjunto de etapas ou processos necessários para que tal projeto seja considerado finalizado para determinada versão, alcançando assim um *milestone*⁴, definido em um sistema de *Issue Tracker*.

³ São funcionalidades do sistema que precisam ser implementadas em determinado momento.

⁴ Determinado ponto do desenvolvimento em que um conjunto de funcionalidades é implementado.

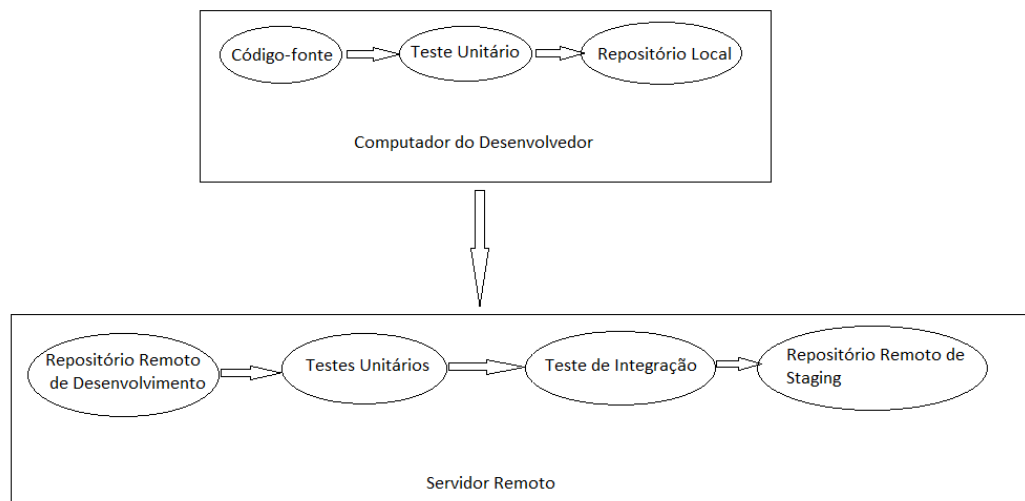


Figura 1 - Exemplo de Etapas de Desenvolvimento utilizando Servidor Remoto

Na Figura 1 pode-se observar um exemplo comum de etapas de desenvolvimento utilizando um servidor remoto para realizar testes unitários e de integração. Estes testes somente são possíveis utilizando ferramentas de automatização de testes de Integração. Pode-se observar também que as alterações locais do desenvolvedor somente são compartilhadas para outros desenvolvedores após terem passadas em um conjunto de testes unitários.

Outra observação pertinente, também referente à importância de uma boa escolha no ambiente de desenvolvimento, é que a criação de vários repositórios que interagem entre si somente é possível com a utilização de ferramentas de Controle de Versão Distribuídas, como Mercurial e Git. Sendo assim, certas ferramentas da mesma categoria, como Subversion, que não proveem esta funcionalidade, não podem ser utilizadas ou então a configuração da etapa de desenvolvimento deve ser alterada para se adequar ao ambiente.

2.4 Modelo Software-as-a-Service e Arquitetura REST

Há diversos modelos de negócios de *software* definidos pela maneira que são distribuídos e comercializados. Quando o *software* é vendido diretamente, sob licença ou entregue a cliente que o encomendou, é dito que este *software* é um produto. Quando não é entregue diretamente, o fornecedor se responsabiliza por toda a estrutura necessária para a disponibilização do sistema (servidores, conectividade, cuidados com segurança da informação) e o cliente o utiliza via

internet, pagando um valor recorrente pelo uso, este *software* na verdade se trata de um serviço ou Software-as-a-Service (SaaS).

Paralelamente, temos o conceito de Web Service, que diz respeito a modularização, reaproveitamento e integração de processos de sistemas heterogêneos. O Web Service pode ser disponibilizado a outros desenvolvedores, aumentando o nicho do negócio da aplicação.

A combinação deste dois modelos vem se provando rentável com sistemas como *Facebook* ou *Twitter*, no qual o desenvolvedor do *web service* fornece chaves para que determinadas aplicações de terceiros possam alimentar o sistema original em troca da possibilidade de consumir seus dados.

Em termos de design de *software*, é vantajoso possuir um *web service*, centralizando a regra de negócio enquanto diversas aplicações consomem seus dados, isto por que tal arquitetura permite a construção de sistemas altamente versáteis e heterogêneos, escaláveis e multi-plataforma, uma vez que ao surgir a necessidade de uma aplicação para determinada plataforma, essa aplicação pode ser acoplada ao sistema original, consumindo os recursos do *web service* central, sem a necessidade de reimplementar a regra de negócio, ao mesmo tempo que os dados do usuário permanecem disponíveis a ele em todos os dispositivos suportados.

Porém, existem diversas arquiteturas possíveis para um *web service*, cada uma delas levando em conta aspectos distintos. Uma dessas arquiteturas é a *Representational State Transfer* (REST), desenvolvido como um modelo abstrato da arquitetura Web, que tenta minimizar a latência e a comunicação na rede, enquanto a mesmo tempo maximiza a independência e a escalabilidade da implementação dos componentes (FIELDING e TAYLOR, 2002).

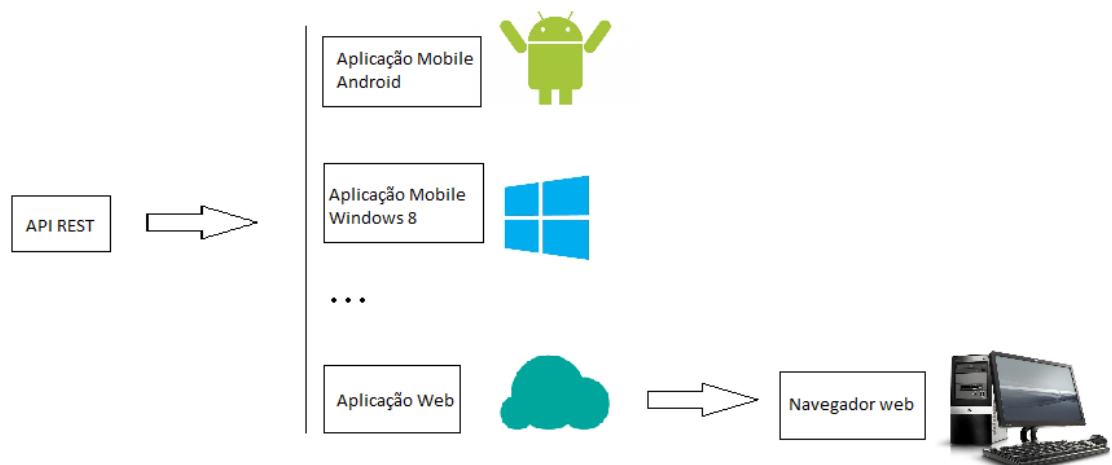


Figura 2 - Utilização de Arquitetura Web Services REST e modelo Software-as-a-Service

Na Figura 2, pode-se analisar o exemplo de uma arquitetura baseada em *web services* utilizando modelo SaaS, no qual o fornecedor do *software* se encarrega de toda infraestrutura, deixando ao seu cliente somente a questão relativa ao seu negócio propriamente dito.

2.5 Desenvolvimento Ágil

Desenvolvimento de Software Ágil teve início em 2001 com a criação do Manifesto do Software Ágil, um documento virtual que é a linha desenhada na areia por dezessete homens trabalhando para encontrar maneiras melhores de criar *software* (BECK, BEEDLE, *et al.*, 2001).

A programação Ágil elimina documentações e processos desnecessários do desenvolvimento formal de software e o substitui com intenção com o cliente, código funcional, antecipação e respostas favoráveis a mudanças, e respeito e valorização dos indivíduos envolvidos. Isto para remover os entraves inerentes ao desenvolvimento de software desde seu começo. Desenvolvimento Ágil se baseia em códigos que funcionam e no *feedback* do cliente, para projetar e ter certeza de que as funcionalidades do software sejam adicionadas baseadas no que o cliente realmente precisa e quer, e não somente no que o desenvolvedor pensa.

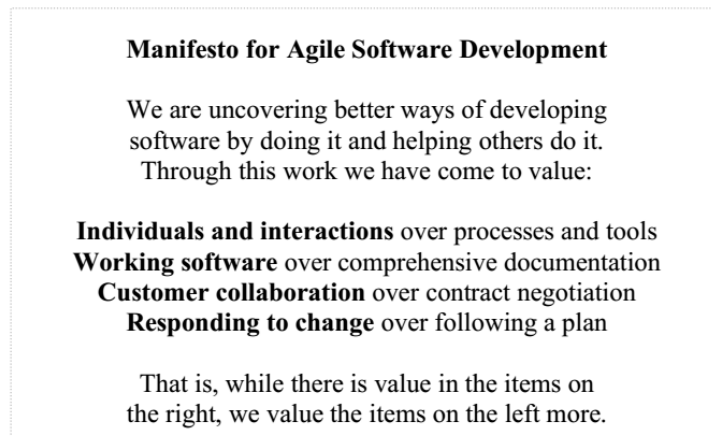


Figura 3 - Manifesto do Software Ágil

Os signatários do Manifesto Ágil (Figura 4), perceberam que podem existir muitos entraves nas metodologias tradicionais de desenvolvimento de software. Por exemplo, requisitos e contratos devem fixados no começo do processo, tornando a mudança impossível, custosa ou

pelo menos, trabalhosa. Também, muito tempo e esforço é gasto criando documentação, diagramas, ou outros artefatos que não são necessários para a entrega final. Então, os métodos Ágeis retira muito dos processos não envolvidos na escrita de software da engenharia de software.

Portanto, tais metodologias clássicas não se adaptam bem à mudanças. Mudanças nos requerimentos iniciais são muitas vezes necessários durante o desenvolvimento, pois estes são extraídos pelos engenheiros de software de maneira incompleta ou mal interpretada. Clientes podem ter se confundido e comunicados suas necessidades imprecisamente, ou as necessidades podem mudar durante o tempo de desenvolvimento. Métodos não-Ágeis tentam lidar com a mudança através de contratos que as eliminam do projeto, não permitindo que novas ou diferentes necessidades sejam consideradas sem renegociação do preço e da data de entrega. Programação Ágil espera e até mesmo adota a mudança como parte normal do desenvolvimento de software. (HOLLAR, 2006)

Metodologias comuns de desenvolvimento ágil incluem: *Extreme Programming*, *Scrum*, Modelamento Ágil, Desenvolvimento de Software Adaptado, *Crystal*, *Lean*, e outros. Mesmo que estas metodologias variem, elas todas possuem em seu núcleo o Manifesto Ágil e permitem que o processo seja customizado para cada necessidade individual de projetos.

2.6 Desenvolvimento Mobile

O desenvolvimento Mobile possui diversas peculiaridades, como potencial integração com outras aplicações, diversas famílias de plataformas de hardware e software, segurança, interface com usuário com elementos comuns com o sistema operacional (design guidelines), complexidade de teste, consumo de energia e grande quantidade de sensores (WASSERMAN, 2010).

Portanto, as soluções de desenvolvimento de Software Ágil podem parecer prover uma boa aplicação no desenvolvimento de aplicações mobile. Porém, as características dos telefones mobile e ambientes de rede colocam algumas restrições na aplicação de qualquer método de desenvolvimento de software ágil. Novas abordagem específicas para mobile são necessárias. (ABRAHAMSSON *et al.*, 2004).

Porém, isto não impede que mesmo desenvolvedores individuais sigam processos parecidos com *Scrum* quando desenvolvem suas aplicações mobile (AGRAWAL e WASSERMAN, 2010) e uma série de “boas práticas” que são resultados compartilhados de experiências práticas de desenvolvedores e também dos próprios distribuidores de sistemas e aparelhos.

2.6.1 Mobile-D

Este método foi desenvolvido para suprir a necessidade de um método ágil voltado à realidade mobile e foi baseada em *Extreme Programming* (práticas de desenvolvimento), metodologia *Crystal* (escalabilidade do método) e Processo Uniforme Racional (cobertura de ciclo de vida) (ABRAHAMSSON *et al.*, 2004).

É otimizado para pequenas equipes com menos de dez desenvolvedores e visando entregar uma completamente funcional em menos de 10 semanas.

Um projeto desenvolvido seguindo Mobile-D é dividido em cinco iterações. As fases são Configuração, Núcleo, Núcleo 2, Estabilização e Empacotamento. Cada fase consiste em três diferentes tipos de dias de desenvolvimento: Dia de Planejamento, Dia de Trabalho e dia de Lançamento, e se múltiplos times estiverem trabalhando concorrentemente no mesmo projeto um Dia de Integração também é necessário.

As práticas das diferentes fases se comprometem a nove elementos principais: Fases e Ritmo, Linha de Arquitetura, Desenvolvimento Mobile Test-Driven, Integração Contínua, Programação Pareada, Métricas, Melhoria de Processo Ágil de Software, Foco centrado no Usuário e Off-Site Client.

A maioria destes conceitos são práticas ágeis conhecidas e somente foram especializadas para o desenvolvimento de software mobile. A Linha de Arquitetura é um princípio específico deste método e consiste em adquirir conhecimentos sobre padrões e soluções existentes que se provaram úteis e funcionais em projetos da organização ou de outras aplicações de fora da organização.

2.7 Metodologia Lean

Startups é, obviamente, um conceito norte-americano introduzido no Brasil entre 1996 e 2001, durante o estouro da bolha ponto-com, das empresas de internet. Pode ser definido como um grupo de pessoas trabalhando com uma ideia diferente que, aparentemente, poderia fazer dinheiro. Além disso, "startup" sempre foi sinônimo de iniciar uma empresa e colocá-la em funcionamento. (O que é uma startup?, 2010)

Lean Startup ou a Startup Enxuta são um conjunto de práticas para criação de novos negócios de forma ágil, com baixos custos e orientada ao desejo dos clientes. Assim, uma lean startup deve possuir as seguintes características (RIES, 2011):

1. Reduzir ao máximo o tempo de criação de seu produto, focando no desenvolvimento do chamado “Produto Mínimo Viável” ou MVP (originário da sigla em inglês “Minimum Viable Product”).
2. Aperfeiçoar continuamente o produto conforme os resultados obtidos, reduzindo os ciclos de desenvolvimento ao máximo.
3. Testar repetidamente a aceitação do produto junto aos seus usuários, coletando informações que possam auxiliar na sua adequação às necessidades dos clientes.
4. Consumir o mínimo possível de recursos (humanos e financeiros) até encontrar o chamado “product x market fit”, ou seja, o produto que se “encaixa” com as necessidades do mercado.
5. Mudar seu produto/modelo de negócio radicalmente se não tiver aceitação do mesmo. É o chamado “pivotar”.

Assim, o conceito que está por trás do lean startup é de, em vez de gastar muitos recursos (tempo, dinheiro e gente) planejando, pesquisando e desenvolvendo, buscar descobrir o que o mercado deseja, testando o mais cedo possível sua aceitação ao produto desenvolvido.

2.8 Experimentação e Exemplificação na Engenharia de Software

As práticas ensinadas pela Engenharia de Software são essenciais para aqueles que pretendem, ao se formar, atuarem como desenvolvedores de software independentes, pois não contam com modelos pré-existentes de uma organização para se basearem. Seus conhecimentos, portanto, se restringem ao que foi aprendido durante cursos, aulas ou até mesmo independentemente. Porém, este aprendizado se dá através de disciplinas teóricas, com aulas expositivas e leituras complementares, que contemplam a competência prática mediante o desenvolvimento

de um pequeno projeto em um curto espaço de tempo (HUANG e DISTANTE, 2006). Deve-se, também, considerar alguns problemas no ensino e estabelecimento da Engenharia de Software como disciplina acadêmica: muitas instituições apresentam carências em prover experiências industriais para os estudantes na prática de Engenharia de Software e também existe a dificuldade de como preparar os estudantes para a prática profissional em um ambiente acadêmico. E como consequência muitos estudantes concluem seu curso de graduação sem terem participado de um projeto (e processos) de Engenharia de Software próximo ao que encontrarão no mundo real (SANTOS *et al.*, 2008).

Porém, quando observamos o contexto atual do desenvolvimento de software, é possível afirmar que mesmo estudantes podem desenvolver e distribuir aplicações mobile durante o período da faculdade e que isto serviria como ponto de partida para a resolução de um desafio de tornar cursos de Engenharia de Software mais atrativos aos estudantes (SANTOS *et al.*, 2008) e estes por sua vez, poderiam possuir experiência com desenvolvimento mobile e engenharia de software, além de expor a importância da disciplinas aos estudantes e fomentar o espírito empreendedor.

Os projetos desenvolvidos por estudantes, então, se tornariam exemplos e casos de uso para futuras turmas, criando um ciclo de aprendizagem e ensino.

3 FERRAMENTAS

Ferramentas básicas são utilizadas constantemente no desenvolvimento de software, desde o seu princípio. Programas como compiladores, montadores e editores de texto, hoje, não são nem considerados ferramentas em si, pois são simplesmente inerentes ao desenvolvimento de software como concebido hoje. Porém, tais programas nem sempre foram considerados componentes essenciais e sim, ferramentas utilizadas para facilitar o trabalho do programador da época.

Um dos princípios básicos deste trabalho é listar ferramentas mais modernas e cuja a função, em um futuro, possa ser considerada tão básica ao desenvolvimento de software quanto as citadas anteriormente, por serem cada vez mais necessárias no dia-a-dia do desenvolvedor.

Para a elaboração deste trabalho foram utilizadas diversas ferramentas e muitas outras foram consideradas, estudadas, mas descartadas por alguma razão. Para que seja possível reproduzir e entender os resultados, é necessário listar e comentar brevemente as ferramentas estudadas, o motivo de sua escolha e até mesmo o motivo de sua rejeição.

3.1 Ambiente de Desenvolvimento

Utilizadas:

- Eclipse: IDE com suporte a vários *plugins* e integrações com outras ferramentas como Maven e Mercurial, ótimas ferramentas de refatoração.
- Visual Studio: IDE padrão para criação de aplicações para plataformas Microsoft, menos *plugins*, mas melhor edição de código, ótimas ferramentas de refatoração.
- Maven: automação de tarefas como build e controle de dependências, aceita como padrão para Java, grande integração com outras ferramentas de Integração Contínua em geral.
- Nuget: controle de dependências para plataformas Microsoft.
- Mercurial: Ferramenta de controle de versão distribuída, possui suporte a sub-repositórios e ótima ferramenta de resolução de conflitos.
- Hudson: Aplicação web de Integração Contínua, suporte a vários *plugins* e integração com Maven e Mercurial.

- Rhodecode: Aplicação web utilizada para criar repositórios Mercurial e controlar seus acessos.
- MySQL: Banco de dados, fácil instalação, poderosas ferramentas de modelagem. Porém, com futuro incerto e inutilizável em produção.
- Tomcat: Servidor de aplicações Java EE padrão.
- Bugzilla: Issue Tracker, contém vários plugins e integração com o Eclipse.

Recusadas:

- Git: Considerado padrão no controle de versão distribuído, porém, não possui suporte a subrepositórios e poucos plugins.
- Jenkins: Ferramenta de Integração Contínua derivada do Hudson, ótima alternativa, porém, de configuração mais complicada.
- NetBeans: Pouca integração com outras ferramentas e serviços.

3.2 Ambiente de Homologação

Utilizadas:

- VMWare Player: Ferramenta de virtualização gratuita, simples de usar com vários drivers para otimizações em vários sistemas, Linux inclusive e especificamente Arch Linux, e versão paga completa, caso necessário.
- Hyper-V: Necessário para virtualizar a plataforma Windows Phone 8.
- Android Virtual Devices: Necessário para virtualizar a plataforma Android.

Recusadas:

- VirtualBox: Pouco suporte e pouca compatibilidade. Porém, o mais completo dos gratuitos. Se o projeto não pretender evoluir a ponto de necessitar de uma virtualização mais completa, este é um bom candidato.

3.3 Ambiente de Produção

Utilizadas:

- DigitalOcean: Serviço de criação de Virtual Private Server, barata e extremamente fácil de se utilizar. Pode-se construir um serviço de cloud de baixa potência mas grande escalabilidade em pouco tempo e com poucos recursos.
- MariaDB: Derivado do MySQL, permite utilizar as ferramentas do mesmo pois mantém a compatibilidade. Mantido pela comunidade open source, portanto, com futuro não ameaçado imediatamente.
- Tomcat: Servidor de aplicações Java EE padrão, leve, gratuito e fácil configuração.
- Apache: Servidor web aceito como padrão, suporte a vários plugins e de configuração bem documentada.

Recusadas:

- Azure: Serviço de cloud da Microsoft. Caro e ainda em constante mudanças, inviabilizando uma estrutura estável.
- Amazon: Caro, mas estável.
- Nginx: Servidor web eficaz e leve, porém de difícil configuração e pouca integração.

3.4 Frameworks e Library

Utilizadas:

- VRaptor: Framework base para Java EE, de origem brasileira, com ótima documentação e boa relação Integração-Configuração. Ótimo suporte a REST utilizando Restfulie.
- Hibernate: Object-Relational Model, robusto e com diversas características vantajosas como sincronização em tempo de execução e engenharia reversa.

- Restfulie: Framework RESTful. Nenhum outro framework para Java possui todos as caracterizas de implementação do modelo REST, no momento.
- Spring Security: Adiciona uma camada de Segurança que se integra com outros frameworks. Boa capacidade de integração no geral, e permite a implementação de diretivas de segurança sem dificuldade.
- JSF: Framework utilizado para componentes gráficos em Java EE.
- SQLite for Windows Phone 8: Permite a criação de Bancos de Dados SQLite no Windows Phone 8.
- SQLite for Windows 8: Permite a criação de Bancos de Dados SQLite no Windows 8.
- AndroidAnnotations: Framework base para Android. Facilita do desenvolvimento ao diminuir a quantidade de código a ser escrito, utilizando anotações pré-processadas.
- Srping REST Template: Permite a utilização de chamadas RESTful, incluindo autenticação, no Android.
- Gson: Conversor objetos para json, formato utilizado na troca de dados escolhido para REST, neste projeto. Mantido pelo Google, utilizado tanto no Android quando no Java EE.

Recusadas:

- Spring MVC: Documentação e configuração complexa, embora seja considerado o melhor framework Java disponível atualmente. Possui componentes próprios para tudo, inclusive ORM, o que dificulta a integração.
- ORMLite: Object-Relational Model, rápido e leve, porém, suas anotações são incompatíveis com Android, tornando impossível o reaproveitamento dos Modelos.

4 RESULTADOS

A seguir serão apresentados os resultados desta pesquisa, que incluem todas as práticas, conceitos e a organização do sistema Think, e por fim, o sistema em si e suas características atuais, com suas *apps* separadas por plataforma. Inclui também, *apps* e Subsistemas que ainda não foram completamente desenvolvidos, mas que possuem potencial para tal.

4.1 Características do Processo de Desenvolvimento

Tendo em vista a necessidade de publicar a aplicação em várias lojas de aplicativos e ainda considerar a possibilidade de um sistema web completo, foi necessário reunir e aplicar diversos conceitos de empreendedorismo, particularmente os utilizados pela metodologia de Startup Enxuta, e combiná-los com as metodologias ágeis, para obter uma visão e implementação completa do projeto.

Tal combinação acarretou na hipótese de que startups de desenvolvimento de software possuem suas próprias atividades divididas em três áreas: Empreendedorismo, Projeto e Desenvolvimento, com cada uma dessas áreas ligada a um valor ou plano de atuação, sendo eles: Ideia, Organização e Produção, respectivamente.

Pode-se dizer que o primeiro passo no desenvolvimento de um software é a elaboração da sua função, o problema a ser resolvido, do seu objetivo, a elaboração da “Ideia” que envolve um certo empreendedorismo, tendo em mente sua aplicação comercial.

O segundo passo, portanto, é o projeto que visa a “Organização” e definição de como tal “Ideia” será aplicada, como um software. Isto inclui o levantamento de quais funcionalidades serão implementadas e quem será o responsável por desenvolvê-las, e em quanto tempo estas ficarão prontas. Ou seja, todas as práticas de uma metodologia de gerência de projeto de software.

Por último, o terceiro passo, é o desenvolvimento em si, a “Produção” daquilo que foi idealizado e então planejado, visando a entrega de código fonte. Tal atividade pode ou não seguir uma série de etapas afim de garantir qualidade e diminuir o esforço do desenvolvimento, tais como versionamento e testes.

Partindo, portanto, de um universo mais abstrato para um mais prático. A Figura 4 ilustra estes universos e exemplos de atividades competentes a cada um. Tal conceito será importante

na definição de responsabilidades de cada participante da equipe e na organização de tarefas a serem realizadas, e sua ordem.



Figura 5 - Organização das Etapas do Processo

É importante ressaltar que cada área visa a entrega de um artefato específico, que é composto pelos artefatos entregues na área anterior, e os completando com suas próprias características. A área do Desenvolvimento entregará o código produzido, a área do Projeto entregará uma versão funcional do software, reunindo diversos códigos produzidos por diversos desenvolvedores ao longo de um ou mais ciclos de desenvolvimento, e a área de Empreendimento entregará um produto, que pode ser composto de diversos *softwares* ou apps para plataformas diversas, que podem ter passado por diversas iterações do ciclo de projeto, muitas vezes compondo um sistema complexo.

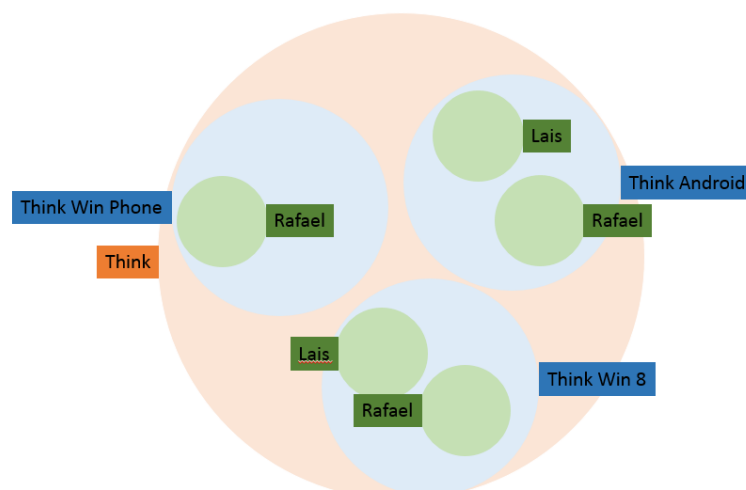


Figura 6 - Exemplo de Organização das Etapas do Processo

Tem-se, como exemplificado na Figura 5, a área do Empreendimento, envolvendo a “Ideia” e suas aplicações, e para a aplicação desta ideia há um ou mais Projetos, para plataformas diversas, envolvendo a “Organização” das tarefas que representam funcionalidades a serem implementadas, e para cada projeto existe o Desenvolvimento de tais funcionalidades, associado à um desenvolvedor, que visa a “Produção” de código, imagem, texto ou qualquer outro componente necessário para a implementação da funcionalidade.

Uma característica interessante desta visão do processo de desenvolvimento é que cada área é cíclica (Figura 6), e que a cada iteração de uma área compõe um passo para a iteração da área superior. Ou seja, a cada ciclo de desenvolvimento, alcança-se a implementação de uma ou mais funcionalidades, que nada mais é do que uma pequena parte da *sprint*. A cada ciclo de projeto, entrega-se uma *sprint*, ou seja, uma versão funcional do *software* que implementa um conjunto de funcionalidades, e portanto, é uma pequena parte do que precisa ser realizado para alcançar a realização da ideia, que para a metodologia Enxuta é chamado de Pivô.

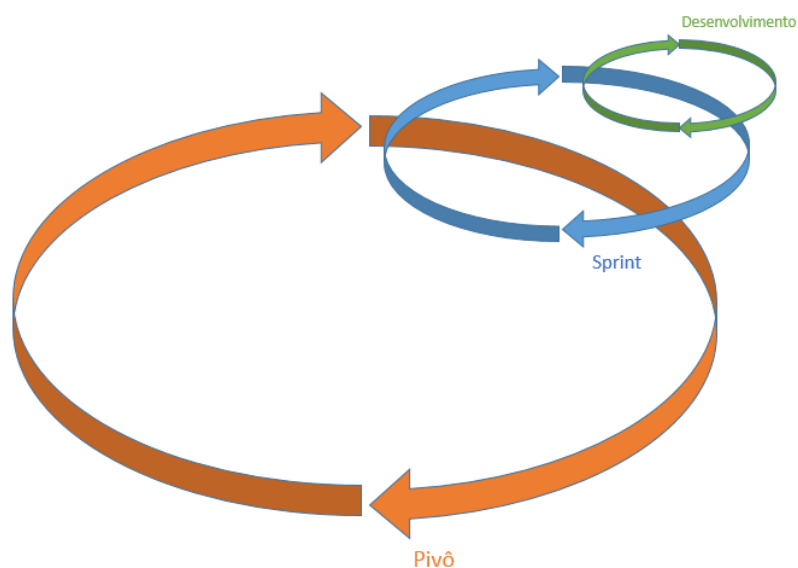


Figura 7 - Processos orientados a Ciclos

Esta característica é importante se mostrou importante por dois motivos, sendo o primeiro dele o acompanhamento do estado do projeto de acordo com seu pivô, sprint e revisão. Com estas informações é possível estimar o esforço empregado pela equipe e determinar a maturidade da aplicação da ideia. Um pivô com muitas *sprint's*, seguido de um pivotamento pode

significar uma abordagem errada por parte da equipe, o que pode ser considerado na próxima iteração.

O segundo motivo é a liberdade que cada ciclo apresenta dentro da mesma área. Cada projeto para cada plataforma pode avançar independentemente, desde que contribua para o pivô, assim como cada desenvolvedor pode trabalhar independentemente contanto que contribua para a evolução da sprint. Isto garante que os investimentos de esforços possam ser realizados nos projetos corretos, dada a adesão para determinada plataforma.

4.2 Empreendimento: A Ideia

A Ideia básica do sistema Think é prover citações famosas para os usuários, através de aplicações mobile, que os façam pensar sobre temas diversos, chamados de Categorias. Foi realizado um MVP, desenvolvido em apenas um dia, para Windows 8 que comprovou o interesse da aplicação para a plataforma, alcançando mil downloads em uma semana.

Após diversos *sprint's* a aplicação conta com mais de 8 mil usuários ativos na plataforma e 30 mil downloads (Figura 7).



Figura 8 - Think publicado na Windows Store

Diversas funcionalidades foram adicionadas com base nos pedidos dos usuários como suporte a internacionalização, tanto da aplicação quanto de seu conteúdo. Houve também grande interesse em participar ativamente da produção de conteúdo e de tradução da aplicação para diversos idiomas.

Para o Android, a aplicação teve pouco destaque. Contanto com aproximadamente 100 downloads em um período de um mês. Ao se analisar o motivo através de ferramentas disponibilizadas pelo Google para analisar dados de acesso à aplicação, observou-se que poucos usuários conseguiam encontrar a aplicação pesquisando termos-chave como Citações. Seria

necessário portanto, uma divulgação maior das aplicações utilizando serviços de “review” de aplicações, que contribuem para a popularização da aplicação na internet e portanto para o próprio algoritmo de relevância do Google.

A aplicação não chegou a ser publicada para Windows Phone 8, porém existiu o interesse dos usuários de Windows 8 pela aplicação, o que fez com que o MVP da aplicação fosse terminado.

Infelizmente, o Desenvolvimento Contínuo para a plataforma *mobile* é motivo de conflito, pois o usuário precisa ser notificado sobre a atualização da aplicação e muitas vezes, autoriza-la, pois atualizações automáticas de aplicações poderiam acarretar em violações de segurança e de privacidade por parte de desenvolvedores maliciosos.

O desenvolvedor honesto é muitas vezes, portanto, obrigado a ignorar usuários com versões muito antigas da aplicação devido a problemas de incompatibilidade entre o web service e a aplicação utilizada pelo usuário, violando assim o princípio de desenvolvimento contínuo.

Foram elaborados diversos planos de negócios, incluindo, resumidamente:

- A. Aplicações gratuitas com limitações e aplicações pagas.
- B. Propaganda comum em todas as aplicações, baseada nas referências utilizadas.⁵
- C. Utilizar a base de dados para gerar produtos, como camisetas, cartões, canecas, etc.
- D. Propagandas sobre os produtos utilizados nas referências.⁶

4.2.1 Multiplataforma

Algumas observações devem ser feitas quando o empreendimento visa, em algum momento, se tornar multiplataforma, para aumentar suas chances de sucesso.

O sistema deve ser levado em consideração. Isto significa que determinadas características do sistema podem e irão influenciar a aplicação desenvolvida para ele, normalmente visualmente. Para isto, na maioria dos casos, existem uma série de regras estabelecidas pela própria empresa que mantém a plataforma, chamadas de *Design Guidelines*, afim de garantir uma experiência fluida para o usuário, diminuindo a curva de aprendizado do usuário. Não seguir estas regras pode culminar na não publicação da aplicação ou na pior das hipóteses, em uma má experiência para o usuário.

⁵ Exemplo: o nome do autor é um link que leva a um site de compra para livros daquele autor

⁶ Exemplo: livros de um determinado autor ganham destaque nas citações entregues diariamente

O perfil do usuário da plataforma também deve ser levado em consideração. Usuários de uma determinada plataforma podem ser mais relutantes em comprar a aplicação, enquanto para outras aplicações com propaganda podem ser extremamente mal vistas.

Segundo o perfil de diferentes usuários, foi estabelecido a seguinte relação de tipo de aplicação por plataforma:

- **iOS:** App paga e app gratuita com limitações (como a quantidade de citações)
- **Android:** App gratuita com propagandas e limitações, e app paga
- **Windows 8:** App paga (com período de testes de 30 dias)
- **REST API (B2B):** Chave de utilização paga, sem limite de uso

4.3 Gerencia do Projeto: A Organização

Para gerenciar o projeto foi necessário adaptar algumas metodologias ágeis existentes, devido as seguintes características da equipe e do projeto: a equipe é reduzida e o projeto é multiplataforma.

Definiu-se uma equipe reduzida como uma equipe na qual as funções serão obrigatoriamente acumuladas por algum integrante. Ou seja, um ou mais membros da equipe terão funções além daquela que é sua área de atuação, como por exemplo, um desenvolvedor atuando como testador (*tester*). Uma equipe pode ser reduzida por diversos motivos, como no caso de uma startup com recursos limitados, ou como no caso de um grupo de colegas trabalhando informalmente em uma aplicação durante o tempo livre. Algumas metodologias ágeis eliminam o papel do gerente de projetos, como a *Extreme Programming*, enquanto outras são voltadas para desenvolvedores individuais, como a *Cowboy*, e outras ainda são adaptadas de metodologias clássicas e permitem a criação de cargos e funções, como o *Scrum*, ainda que horizontalmente.

Porém, nenhuma delas analisa a aplicação de esforço e a distribuição de tarefas para membros que não necessariamente são especializados para elas, ou determinam como tais funções serão exercidas. Utilizando o conceito de equipes reduzidas, é possível analisar a equipe e distribuir funções de maneira a minimizar o impacto sobre seus membros, quando tal impacto não pode ser evitado.

Portanto, é possível que tal equipe seja funcional e eficiente, desde que suas funções sejam compatíveis com suas especializações originais. Para tal, parte-se das seguintes suposições

iniciais: as especializações iniciais dos membros da equipe serão de Desenvolvedor (programador) e Designer.

Assim, são cobertas as funções básicas do ciclo de desenvolvimento, que seriam gerar código e gerar *assets*⁷. Obviamente, o desenvolvedor realiza muito mais do que gerar código, pois existe a preocupação com a estrutura do programa, seu desempenho, sua manutenibilidade, e a segurança, existindo portanto um planejamento de tal trabalho. O mesmo é aplicado ao designer, pois antes de gerar *assets*, é necessário visualizá-los como um todo através de um protótipo e analisar a experiência que o usuário teria utilizando tal composição visual do programa.

Porém, em uma estrutura hierárquica de desenvolvimento, o programador é relegado exclusivamente à programação, deixando a estrutura do código aos Analistas de *Software*, e o mesmo pode ser dito aos designers que teriam, originalmente, duas funções distintas, a de elaborar a interface do *software*, determinando a posição e a função dos componentes visuais (designer de interface) e a de determinar a aparência destes componentes, como cores e formas, integrando a identidade visual do sistema (designer visual).

Porém, como as funções são compatíveis entre si em ambos os casos, é aceito que estas possam se acumular, principalmente em equipes Ágeis, que tendem a serem menores do que equipes tradicionais. O mesmo, ainda, pode ser aplicado à outras funções além destas, levando em consideração a expertise dos membros da equipe. As funções comuns em uma equipe de desenvolvimento de *software* são:

- Desenvolvedor
- Administrador de Sistemas
- Gerente de Projetos (ou *Scrum Master*)
- Designer
- Tester
- Product Owner

Como algumas metodologias ágeis já desconsideram a função de gerente de projeto, foi estabelecido no escopo deste projeto que uma equipe reduzida portanto é aquela que possui

⁷ Assets são componentes visuais, textuais e sonoros que não são codificados, e sim, incluídos no projeto em formato binário, como imagens, fontes, efeitos sonoros, etc.

menos do que cinco integrantes e, portanto, passíveis de deficiências ou de acúmulo de funções por parte de seus membros. Uma equipe reduzida pode possuir mais do que cinco integrantes desde que existam funções acumuladas entre seus membros, portanto, por exemplo, uma equipe onde todos membros tem responsabilidades em todas as funções do projeto, sem a definição clara de função para os especialistas (ou na ausência de especialistas), é dita reduzida independente de seu tamanho.

4.3.1 Scrum.M Reduzido

Para tal situação, e considerando ainda o contexto multiplataforma, no qual a equipe será obrigada a lidar com vários ambientes diferentes, foi estabelecido um diverso conjunto de práticas, reunidos de outros métodos ágeis, principalmente do *Scrum*. Esta metodologia não chega, portanto, a ser uma metodologia e sim somente uma adaptação de metodologias existentes, sendo elas, o *Scrum*, no qual foi baseada a estrutura básica, o Cowboy, do qual foram tirados os conceitos de automatização, e o Mobile-D, que possui características para lidar com o desenvolvimento mobile.

Esta adaptação utiliza todos os conceitos do *Scrum*, que incluem a utilização de Backlogs e de User Stories e, como consequência, a utilização de todos os gráficos necessários. Além disso, conceitos como Test Driven Development e Métricas, Foco Centrado no Usuário e Integração Contínua são utilizados baseados na metodologia Mobile-D.

As novidades estão na definição de Testes e Documentação automatizados, utilizando ferramentas que os gerem automaticamente, baseados em comentários. Isto porque elimina esforço voltado para estas tarefas se fossem feitas manualmente. Em especial, os testes unitários e de integração são escritos utilizando Test Driven Development, que exige do desenvolvedor definir os testes antes de escrever os códigos, delimitando o domínio das funções e definindo seus cabeçalhos antes mesmo de escreve-las, determinado seu comportamento, até mesmo quando ocorre um erro.

Outra novidade é ausência de *Scrum Master*, com o comprometimento de todos da equipe cumprirem esta função. Tal exigência é necessária dada a falta de um membro especializado, porém perigosa, todos devem garantir o bom andamento do projeto e propor levantar User Stories (com exceção daquele que estiver efetuando o papel de Product Owner). O Product Owner pode ser acumulado por um designer, preferencialmente. O contato do designer com o cliente é mais próximo que o do desenvolvedor, pois o designer precisa se focar nas necessidades e nas features que serão implementadas como um todo, para a elaboração do protótipo. Tal protótipo

é uma ferramenta utilizada para dar à todos os stakeholders uma visão de como a aplicação ficará em seu sprint atual, auxiliando em testes de usabilidade. Porém, pode ser facilmente utilizado como uma representação visual do Backlog da sprint.

As reuniões devem ser evitadas e a comunicação deve ser feita de maneira instantânea entre todos os membros da equipe. Pouco membros permitem uma comunicação mais clara e rápida sobre o estado do projeto.

Existem também dois conceitos além daqueles estabelecidos pela própria filosofia ágil. Reaproveitamento Extremo, que visa o reaproveitamento através da utilização de frameworks e da separação de código que possa ser reutilizado e Aproveitamento Extremo, visando utilizar todas as ferramentas disponíveis pelo sistema para qual se está desenvolvendo, como é o caso de ferramentas de compartilhamento em redes sociais e componentes visuais.

A iteração, portanto, pode ser estabelecida da seguinte forma, utilizando como base a iteração fornecida pelo próprio *Scrum*, com o tempo entre cada iteração variando de uma a três semanas, por plataforma:

Validação (ou criação) do Backlog pelo Product Owner

Planejamento: Levantamento de User Stories que serão implementadas

Elaboração Protótipo

Desenvolvimento (Working): de Código e de Assets

Levantamento de métricas e atualização do Backlog.

Entrega

4.4 Desenvolvimento: A Produção

A fase de desenvolvimento, ou working, definida pela adaptação Scrum.M Reduzido é orientada a Etapas e à Ferramentas. Isto significa que é impossível se referir ao desenvolvimento e suas etapas sem se referir as ferramentas utilizadas.

4.4.1 Etapas de Desenvolvimento

As etapas de desenvolvimento são a menor unidade no processo de desenvolvimento que podem ser gerenciadas, e portanto, se referem a ações simples como “escrever código” ou “realizar testes” ou ainda “consolidar revisão de código”.

Portanto, estas etapas são fortemente ligadas à estrutura do ambiente de desenvolvimento, que por sua vez é composto com uma série de ferramentas, e estas ferramentas são o que tornam possível a execução destas etapas, como o versionamento. Só é possível realizar um versionamento prático utilizando uma ferramenta de versionamento, como por exemplo, o Mercurial, e só é possível de maneira distribuída utilizando ferramentas para este propósito, e não outro. Assim, uma etapa de versionamento utilizando Mercurial, é completamente diferente do versionamento utilizando Subversion, que é outra ferramenta de versionamento.

E a utilização ou não de uma determinada ferramenta pode modificar drasticamente o que é possível de se fazer nas etapas de desenvolvimento. Abaixo, a Figura 8 descreve um possível ciclo automatizado de testes, que só é possível utilizando controle de versão distribuído. Na figura, os triângulos são ambientes, os retângulos são repositórios de arquivos e os círculos são ferramentas, e as setas são ações, automatizadas ou não.

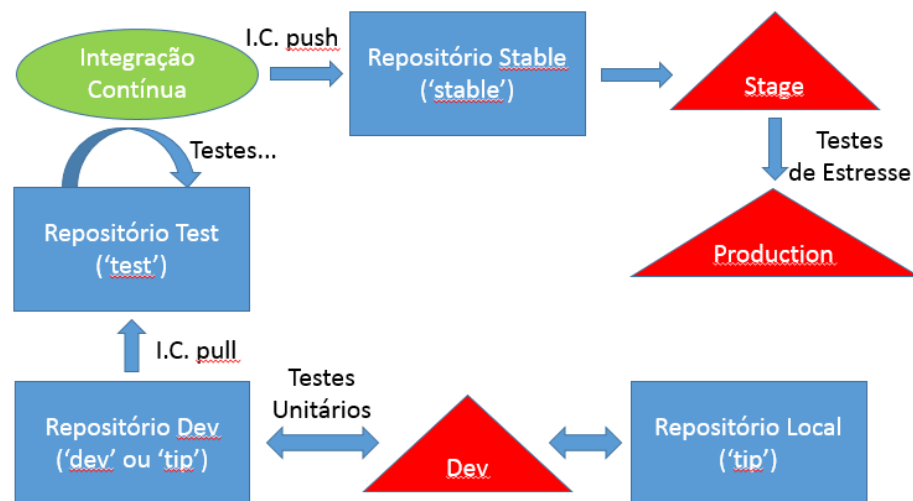


Figura 9 - Exemplo do versionamento nas etapas de desenvolvimento

A lista a seguir exemplifica etapas de desenvolvimentos e ferramentas que podem ser utilizadas. Seguindo o conceito de processo orientado a ciclo, um desenvolvedor executaria todas estas etapas, na ordem proposta, em um período necessariamente menor que o da sprint, como em um dia por exemplo.

- **Features e Bugs:** *Mylyn*
- **Desenvolvimento:** *IDE e gerenciamento de dependência como nuget e maven*
- **Teste Unitário Automatizado:** *Frameworks de Teste Unitário em geral*

- **Documentação Automatizada:** *Javadoc e Doxygen*
- **Versionamento:** *Mercurial ou Git*
- **Integração Automatizada:** *Hudson ou Jenkins*
- **Bug reports e feature requests:** *Bugzilla, Fly Spray, Phabricator, Jira*

4.4.2 Aplicação de Esforço

A aplicação de esforço é importante desenvolvimento em equipes reduzidas pois evita que o desenvolvedor perca o foco do que realmente deve ser feito. A Figura 9 ilustra como ao se dedicar muito à uma determinada características, perde-se tempo que poderia ser utilizado nos demais. Estes tipos de componentes são auto exclusivos e é trabalhoso mantê-los maximizados.

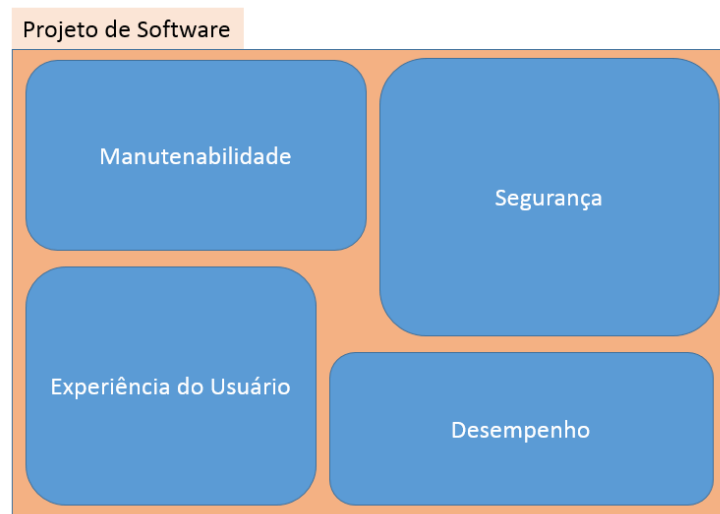


Figura 10 - A aplicação de esforço nas características de um projeto de Software

Durante o desenvolvimento do Think, tentou-se manter todas as características maximizadas, o que se mostrou quase impossível. Ao se melhorar a experiência do usuário, por exemplo, tornando os componentes visuais mais importantes, perdia-se a manutenção pois o código desses componentes fugia do padrão dos demais, o que gerava a criação de novas interfaces. Também se perdia em desempenho, pois tais objetos possuíam mais código ineficiente e também segurança pois se salva a senha para que o usuário tenha a comodidade de não ter que digitá-la sempre. Da mesma maneira, ao se aumentar a segurança, como outro exemplo, era necessário adaptar as classes que precisavam de segurança no acesso e essas se tornavam as

que precisavam de novas interfaces, ferindo a manutenibilidade. Obviamente, a criptografia prejudicava o desempenho e o usuário era prejudicado ao ser necessário se autenticar.

Desta maneira, recomenda-se a utilização moderada das quatro características e utilizar o esforço onde for necessário, somente.

4.4.3 Organização de Código

A organização do código-fonte é de extra importância em um projeto multiplataforma, pois a não organização pode levar a queda da qualidade do *software*, ou em retrabalhos desnecessários, e em casos extremos, na perda de código através de exclusões acidentais.

Para evitar tais inconvenientes e otimizar o versionamento e ainda fazer uso do Reapro-
veitamento Extremo, pode-se criar a seguinte estrutura de pastas:

Nome do Produto, exemplo: Think

Documentação, relativos ao produto

Binários, binários para todas as plataformas

Pasta para Plataforma, exemplo: Think-Web

Pasta para Projeto, exemplo: think-rest

Pasta para Projeto, exemplo: think-website

Pasta para Library, exemplo: think-core

Pasta para Plataforma, exemplo: Think-Android

Pasta para Projeto, exemplo: think-android

Pasta para Projeto, exemplo: think-android-it

Pasta para Library, exemplo: think-core⁸

Pasta para Plataforma, exemplo: Think-Windows

Pasta para Projeto, exemplo: Think-WinStore

Pasta para Projeto, exemplo: Think-WinPhone

Pasta para Library, exemplo: Think-Core⁹

⁸ Neste caso, pode se criar um link entre para a pasta think-core original

⁹ É difícil criar bibliotecas que possam ser utilizadas por plataformas diferentes, portanto, uma biblioteca por plataforma é necessária em muitas situações.

Utilizando o Mercurial, é possível versionar cada projeto de cada plataforma e utilizar o conceito de subrepositórios para versionar o diretório da plataforma inteiro, incluindo os projetos como subrepositório. É possível utilizar o conceito de Reaproveitamento Extremo criando links para projetos de plataformas diferentes mas que podem ser reaproveitados, como é o caso do think-core, para Android e web baseado em Java. Também é possível manter as estruturas originais para cada projeto, para evitar problemas de compatibilidade com a IDE ou com ferramentas de controle de dependência e de automatização de tarefas, como Maven.

4.5 Aplicações e Projetos

Foram desenvolvidas aplicações para diversas plataformas mobile, entre elas:

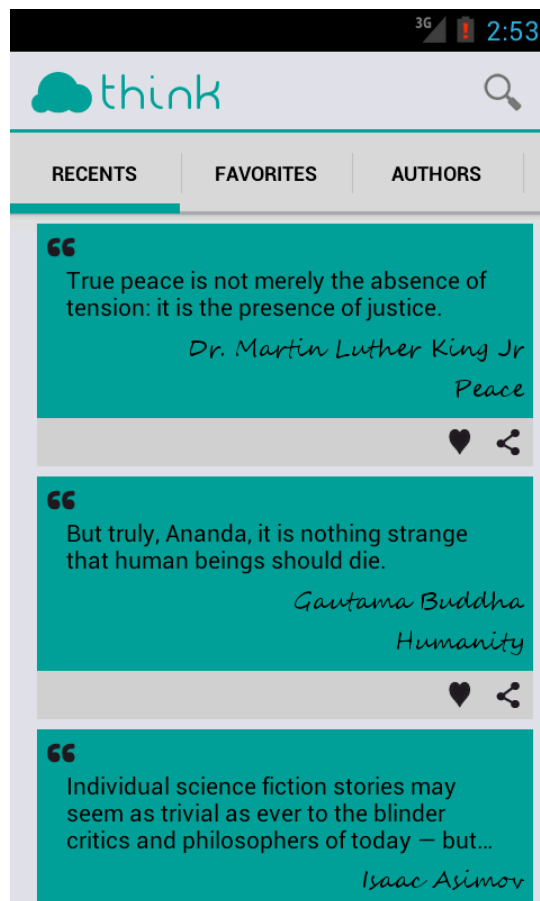


Figura 11 - Think para Android - Smartphone

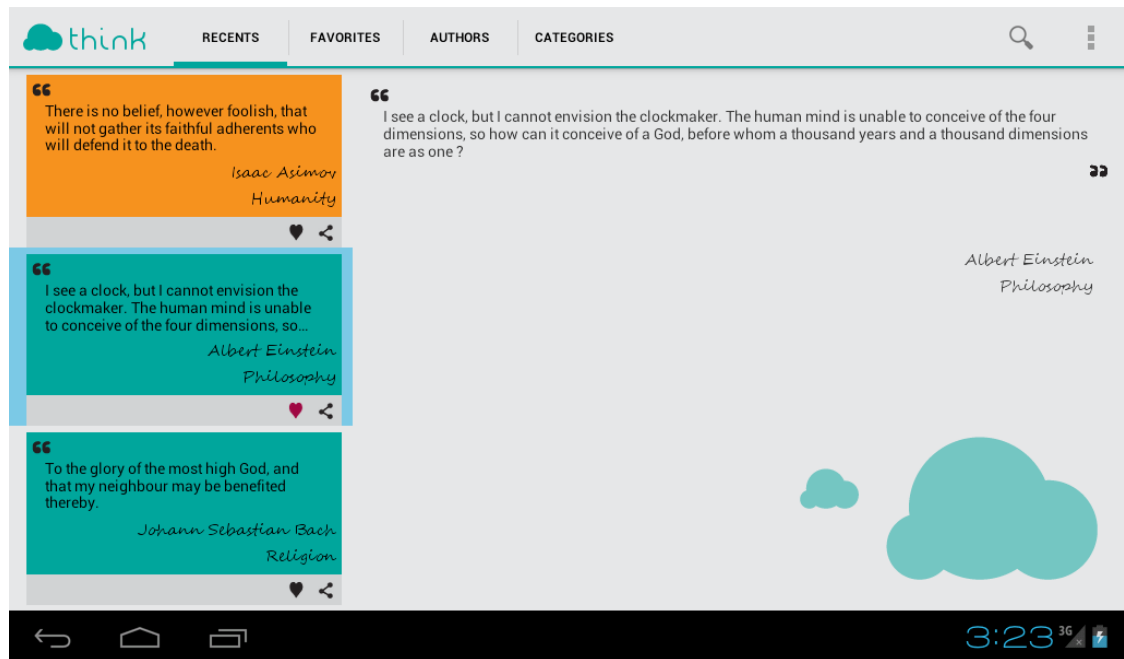


Figura 12 - Think para Android - Tablet

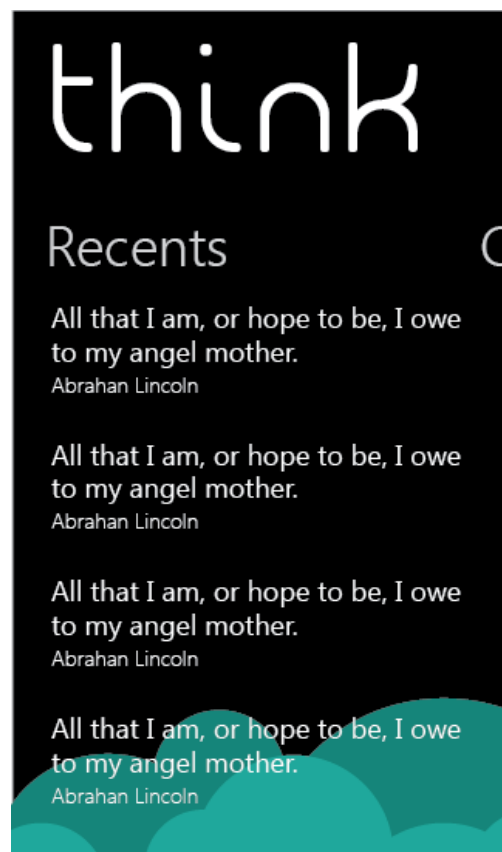


Figura 13 - Think para Windows Phone, MVP



Figura 14 - Think para Windows

Foi utilizado o conceito de Aproveitamento Extremo para Android, ao criar uma aplicação que funcionasse tanto para tablete (Figura 11), quanto para smartphones (Figura 10), utilizando o conceito de *fragments* disponibilizado pelo próprio sistema. Da mesma maneira, o conceito foi aplicado no Windows 8 (Figura 13), Windows Phone 8 (Figura 12) e no Android, para o compartilhamento em Redes Sociais.

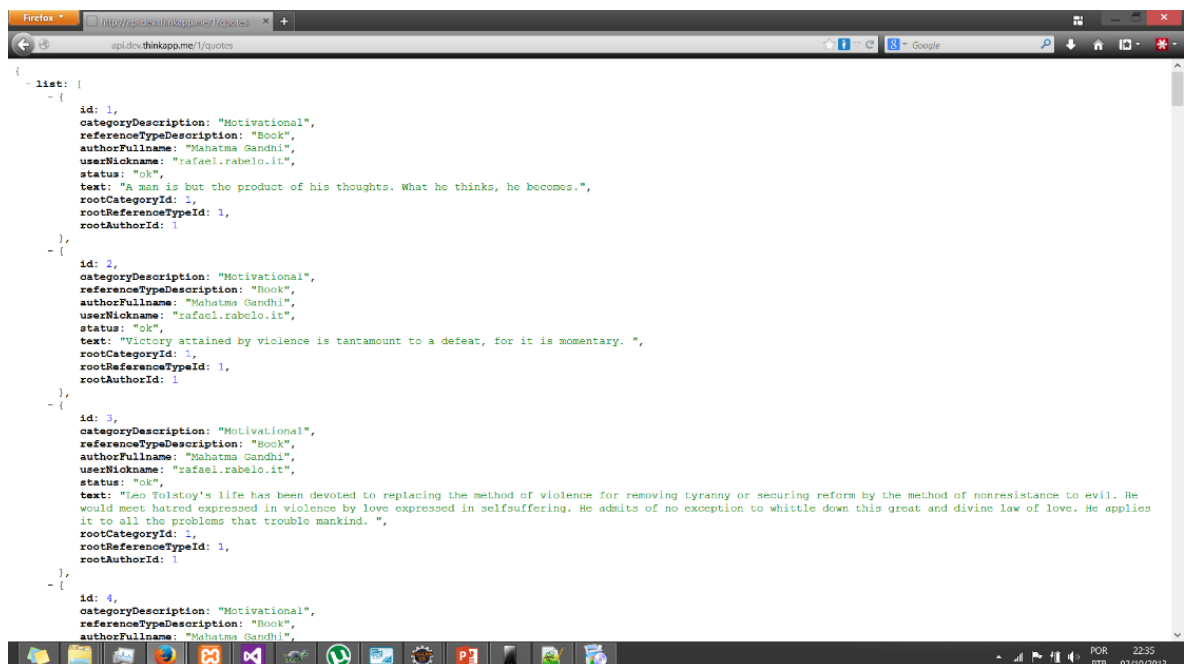
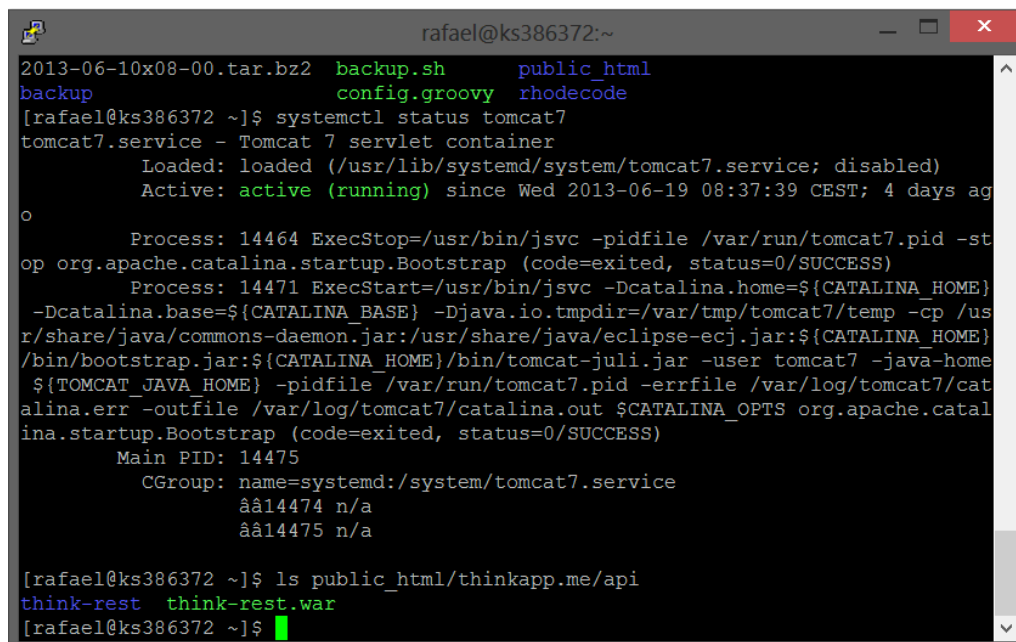


Figura 15 - API REST

O conceito de Reaproveitamento Extremo foi utilizado ao extrair as classes de Modelo e outras que poderiam ser utilizadas em diversas plataformas como Java EE e Android, e Windows 8 e Windows Phone. Também na utilização de padrões tanto de segurança quando de *design patterns*.

O acesso à API REST (Figura 14) é realizado somente utilizando uma chave que é distribuída junta com a aplicação. Tal chave assimétrica assina a requisição e é validada pelo servidor, o que impede a utilização da API por terceiros não autorizados e permite a utilização da mesma para o mercado B2B, vendendo chaves para que outros possam utilizar o serviço e incrementar suas próprias aplicações ou negócios.



```

rafael@ks386372:~
2013-06-10x08-00.tar.bz2  backup.sh  public_html
backup                    config.groovy rhodecode
[rafael@ks386372 ~]$ systemctl status tomcat7
tomcat7.service - Tomcat 7 servlet container
   Loaded: loaded (/usr/lib/systemd/system/tomcat7.service; disabled)
   Active: active (running) since Wed 2013-06-19 08:37:39 CEST; 4 days ago
     Process: 14464 ExecStop=/usr/bin/jsvc -pidfile /var/run/tomcat7.pid -stop
     op org.apache.catalina.startup.Bootstrap (code=exited, status=0/SUCCESS)
     Process: 14471 ExecStart=/usr/bin/jsvc -Dcatalina.home=${CATALINA_HOME}
     -Dcatalina.base=${CATALINA_BASE} -Djava.io.tmpdir=/var/tmp/tomcat7/temp -cp /usr/share/java/commons-daemon.jar:/usr/share/java/eclipse-ecj.jar:${CATALINA_HOME}/bin/bootstrap.jar:${CATALINA_HOME}/bin/tomcat-juli.jar -user tomcat7 -java-home ${TOMCAT_JAVA_HOME} -pidfile /var/run/tomcat7.pid -errfile /var/log/tomcat7/catalina.err -outfile /var/log/tomcat7/catalina.out $CATALINA_OPTS org.apache.catalina.startup.Bootstrap (code=exited, status=0/SUCCESS)
    Main PID: 14475
      CGroup: name=systemd:/system/tomcat7.service
              â€”14474 n/a
              â€”14475 n/a

[rafael@ks386372 ~]$ ls public_html/thinkapp.me/api
think-rest  think-rest.war
[rafael@ks386372 ~]$

```

Figura 16 - Servidor rodando Arch Linux

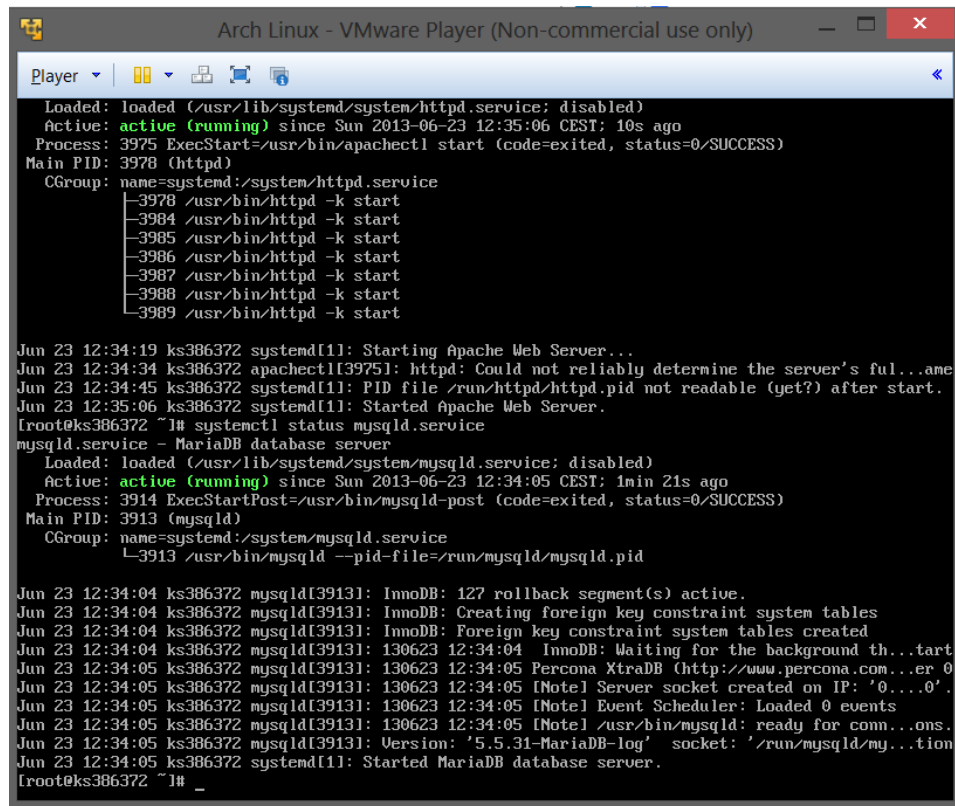
Foram utilizadas máquinas virtuais hospedadas no serviço DigitalOcean, com as seguintes configurações, com o custo de \$ 5,00/mês, acessadas via ssh (Figura 15). Também é possível, se necessário, configurar várias máquinas virtuais para trabalharem em conjunto, como em um ambiente de Cloud Privado.

RAM: 512 MB

CPU: 1 CPU

Armazenamento: 20 GB SSD

Transferência: 1 TB/Mês



```

Arch Linux - VMware Player (Non-commercial use only)

Player ▾  [Icons]

Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
Active: active (running) since Sun 2013-06-23 12:35:06 CEST: 10s ago
Process: 3975 ExecStart=/usr/bin/apachectl start (code=exited, status=0/SUCCESS)
Main PID: 3978 (httpd)
CGroup: name=systemd:/system/httpd.service
├─3978 /usr/bin/httpd -k start
├─3984 /usr/bin/httpd -k start
├─3985 /usr/bin/httpd -k start
├─3986 /usr/bin/httpd -k start
├─3987 /usr/bin/httpd -k start
├─3988 /usr/bin/httpd -k start
└─3989 /usr/bin/httpd -k start

Jun 23 12:34:19 ks386372 systemd[1]: Starting Apache Web Server...
Jun 23 12:34:34 ks386372 apachectl[3975]: httpd: Could not reliably determine the server's full...ame
Jun 23 12:34:45 ks386372 systemd[1]: PID file /run/httpd/httpd.pid not readable (yet?) after start.
Jun 23 12:35:06 ks386372 systemd[1]: Started Apache Web Server.
[root@ks386372 ~]# systemctl status mysqld.service
mysqld.service - MariaDB database server
Loaded: loaded (/usr/lib/systemd/system/mysqld.service; disabled)
Active: active (running) since Sun 2013-06-23 12:34:05 CEST: 1min 21s ago
Process: 3914 ExecStartPost=/usr/bin/mysqld-post (code=exited, status=0/SUCCESS)
Main PID: 3913 (mysqld)
CGroup: name=systemd:/system/mysqld.service
└─3913 /usr/bin/mysqld --pid-file=/run/mysqld/mysqld.pid

Jun 23 12:34:04 ks386372 mysqld[3913]: InnoDB: 127 rollback segment(s) active.
Jun 23 12:34:04 ks386372 mysqld[3913]: InnoDB: Creating foreign key constraint system tables
Jun 23 12:34:04 ks386372 mysqld[3913]: InnoDB: Foreign key constraint system tables created
Jun 23 12:34:04 ks386372 mysqld[3913]: 130623 12:34:04 InnoDB: Waiting for the background th...tart
Jun 23 12:34:05 ks386372 mysqld[3913]: 130623 12:34:05 Percona XtraDB (http://www.percona.com...er 0
Jun 23 12:34:05 ks386372 mysqld[3913]: 130623 12:34:05 [Note] Server socket created on IP: '0...0'.
Jun 23 12:34:05 ks386372 mysqld[3913]: 130623 12:34:05 [Note] Event Scheduler: Loaded 0 events
Jun 23 12:34:05 ks386372 mysqld[3913]: 130623 12:34:05 [Note] /usr/bin/mysqld: ready for conn...ons.
Jun 23 12:34:05 ks386372 mysqld[3913]: Version: '5.5.31-MariaDB-log' socket: '/run/mysqld/my...tion
Jun 23 12:34:05 ks386372 systemd[1]: Started MariaDB database server.
[root@ks386372 ~]# _

```

Figura 17 - Ambiente virtualizado de Homologação

O que pode ser facilmente utilizado localmente, virtualizado (Figura 16), para realizar testes de carga.

Porém, mesmo com um número de 2000 acessos simultâneos realizados localmente, a aplicação continuou responsiva, demonstrando sua eficiência.

Além disso, outros projetos para outras plataformas foram criados, porém, não chegaram a ser implementados, a maioria deles devido à falta de design, devido à falta do design para as interfaces: Aplicação para Facebook; Aplicação para Ginga (TV Digital) e Sticker (TV Digital);

Ao todo, foram gastos 12 meses utilizando o serviço do DigitalOcean (\$ 5,00/Mês) e 1 ano de Assinatura para Windows Store (Windows 8 e Windows Phone, \$ 19,00/Ano) e para Play Store (Android, \$ 25,00/Ano), totalizando \$ 104,00, ou aproximadamente R\$ 210,00, utilizando a cotação dos dias em que os serviços foram adquiridos.

4.6 Guia de Desenvolvimento

O guia de desenvolvimento foi desenvolvido em HTML, no formato de um site simples, para que possa ser atualizado futuramente. O endereço permanente é <http://neptune.li/guia>.

O código fonte do guia também está disponível no github, um serviço de compartilhamento de código fonte aberto, no seguinte endereço <https://github.com/rafaelrabeloit/guiade-senvolvimento>, com o objetivo de aceitar alterações e correções de outros desenvolvedores, tanto em seu conteúdo quanto em seu formato.

Seu formato é, basicamente, o mesmo utilizado neste trabalho, porém de linguagem informal e utilizando mais exemplos, como configurações das ferramentas e tópicos específicos como sobre Maven e Integração Contínua. A estrutura dos tópicos está listada a seguir, com uma pequena descrição para cada tópico:

Introdução

A motivação deste trabalho, citando o estímulo ao desenvolvimento de *software* com qualidade, mesmo durante trabalhos ditos informais, como os de faculdade, utilizando conceitos de Engenharia de *Software*, inclusive com experiências pessoais da faculdade.

Processo de Desenvolvimento Proposto

Descreve as áreas ou universos de atuação, e a motivação de utiliza-las da maneira proposta, orientada a ciclos.

Empreendimento: A Ideia

Orienta ao empreendedor como elaborar a ideia e transforma-la em algo possivelmente rentável e aplicável à um problema real, segundo a metodologia Enxuta, e a como encontrar uma boa ideia.

Gerencia de Projeto: A Organização

Indica ao gerente de projetos como elaborar a iteração, sugerindo *Scrum* como metodologia de gerencia. Também apresenta o Scrum.M reduzido, como abordagem para projetos multiplataforma.

Desenvolvimento: A Produção

Apresenta diversas ferramentas e configurações, como Maven, Mercurial e suas vantagens. Também apresenta as vantagens de se utilizar VPS de baixo custo em projetos pequenos e sua capacidade de escalabilidade.

5 CONCLUSÃO

Com este projeto, foi possível concluir que é possível desenvolver aplicações que atinjam um grande número de usuários mesmo em pequenas equipes, mesmo que o foco da produtividade dos membros estejam em outras tarefas, desde que haja espírito empreendedor, organização e força de vontade. O mundo, hoje, oferece um número enorme de ferramentas para auxiliar o desenvolvedor, e ao fazer uso destas, grande parte de seu trabalho é facilitado e otimizado.

Além disto, há diversas bibliotecas e frameworks de código aberto que, uma vez conhecidos, podem acelerar muito o desenvolvimento de aplicações, inclusive aplicações mobile, e manter uma alta qualidade, uma vez que uma parte considerável destes frameworks são testados adequadamente, se mostrando eficientes, como é o caso do AndroidAnnotations e do VRaptor.

Conclui-se também que, utilizando ferramentas gratuitas e serviços baratos de virtualização disponíveis à qualquer usuário e não só às grandes empresas, é possível criar um *software* de qualidade, escalável e eficiente. Isto, somado ao baixo custo das lojas de aplicativos dá ao desenvolvedor independente capacidade de criar uma aplicação de sucesso em pouquíssimo tempo. Porém tais oportunidades parecem não ser aproveitadas ao máximo nem por respeitáveis empresas brasileiras, que ainda apresentam serviços instáveis e sistemas que carecem de qualidade.

Com o mercado de aplicações móveis cada vez mais importante e relevante, tanto no cenário nacional quanto internacional, são esperados cada vez mais trabalhos voltados para estas plataformas e este projeto pode ajudar tais desenvolvedores a melhorarem a produtividade durante o desenvolvimento destas aplicações. Ainda, foi observado neste trabalho, que o contexto mobile apenas foi utilizado para acrescentar dificuldades e peculiaridades inerentes ao universo multiplataforma, portanto, este trabalho pode facilmente ser adaptado para incluir o desenvolvimento web que, inclusive, acabou sendo feito durante a produção da API REST, comprovando mais uma vez o conceito “mobile first”, que consiste em elaborar o sistema pensando primeiro nas aplicações mobile, e só então nas aplicações web, conceito este defendido por desenvolvedores do mundo inteiro como uma boa-prática essencial.

REFERÊNCIAS

- ABRAHAMSSON, P. et al. **Mobile-D: An Agile Approach for Mobile Application Development**. OOPSLA '04 Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. [S.l.]: [s.n.]. 2004. p. 174-175.
- AGRAWAL, S.; WASSERMAN, A. I. **Mobile Application Development: A Developer Survey**. [S.l.]. 2010.
- BECK, K. et al. Manifesto for Agile Software Development, 2001. Disponível em: <<http://agilemanifesto.org>>. Acesso em: 20 Julho 2013.
- BOSCH, J. et al. Framework - Problems and Experiences. In: FAYAD, M.; SCHMIDT, D.; JOHNSON, R. **Building Application Frameworks: Object-Oriented Foundations of Frameworks Design**. New York: John Wiley, 1999.
- DE LUCENA, V. F.; BRITO, A.; GOHNER, P. A. **A Germany-Brazil Experience Report on Teaching Software Engineering for Electrical Engineering Undergraduate Students**. CONFERENCE ON SE EDUCATION & TRAINING – CSEET '06. [S.l.]: [s.n.]. 2006. p. 69-76.
- FAYAD, M. E.; LAITINEN, M.; WARD, R. P. Thinking objectively: software engineering in the small. **Magazine Communications of the ACM**, v. 43, n. 3, p. 115-118, 2000.
- FIELDING, R. T.; TAYLOR, R. N. Principled Design of the Modern Web Architecture. **ACM Transactions on Internet Technology**, v. 2, n. 2, p. 115-150, 2002.
- HOLLAR, A. B. **Cowboy: An Agile Programming Methodology for a Solo Programmer**. [S.l.]. 2006.
- HUANG, S.; DISTANTE, D. **On Practice-Oriented Software Engineering Education**. Conference on Software Engineering Education & Training Workshops – CSEETW '06. Washington: IEEE Computer Society. 2006. p. 15.
- MOSER, S.; NIERSTRASZ, O. The Effect of Object-Oriented Frameworks on Productivity. **IEEE Computer**, p. 45-51, 1996.
- O que é uma startup? **Exame.com**, 2010. Disponível em: <<http://exame.abril.com.br/pme/noticias/o-que-e-uma-startup>>. Acesso em: 22 Julho 2013.

PRAHALAD, C. K.; HART, S. **The Fortune at the Bottom of the Pyramid**. [S.l.]: [s.n.], 2001.

RIES, E. **The Lean Startup**. [S.l.]: [s.n.], 2011.

ROGERS, G. F. **Framework-Based Software Development in C++**. New Jersey: Prentice Hall, 1997.

SANTOS, R. P. D. et al. **Utilizando Experimentação para Apoiar a Pesquisa em Educação em Engenharia de Software no Brasil**. I FEES - I Fórum de Educação em Engenharia de Software. [S.l.]: [s.n.]. 2008.

SBC. **Grandes Desafios da Pesquisa em Computação no Brasil – 2006-2016**. Relatório da Sociedade Brasileira de Computação. [S.l.], p. 22. 2006.

WASSERMAN, A. I. **Software engineering issues for mobile application development**. 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Santa Fe: [s.n.]. November 2010. p. 397-400.